COMPSCI 720S1C, 2006

Mark C. Wilson

May 16, 2007

Probability basics I

- A probability space is a set X with a probability measure $\Pr$ defined on a σ-algebra $\mathcal{S}$.
- A σ-algebra is a collection of subsets of X that contains $\emptyset$ and is closed under complement, unions.
- A probability measure is a function $\Pr : \mathcal{S} \to [0, 1]$ such that $\Pr(\emptyset) = 0; \Pr(\cup_i A_i) = \sum_i \Pr A_i$ if the A_i are pairwise disjoint.
- For us, usually X is finite of size n and $\mathcal{S}$ contains all the 2^n subsets of X. The space is discrete.
- A random variable is a (measurable, real-valued) function on X. For us, the "measurable" can safely be ignored.

Probability basics II

- The mean of a discrete random variable T is $\mu := E[T] := \sum_{x \in X} T(x) \Pr(\{x\}) = \sum_y y \Pr(T(x) = y)$.
- The variance of T is $\sigma^2 := E[T^2] - E[T]^2$.
- Chebyshev's inequality says that $\Pr(|T - \mu| > c\sigma) < c^{-2}$ for each $c > 0$. In practice this probability is often exponentially small in c.
- The measure is uniform if the probability of a subset depends only on its size. The probability of a k-element subset of an n-element space is then k/n. In this case all the above quantities can be computed via counting.

Recalling ideas from previous courses

- We aim to compare resource use of algorithms for a given computational problem. A mathematical model is needed.
- We measure asymptotic running time for large inputs. Small inputs are not a challenge. We don't care about constant factor speedups due to faster machine, better programming, etc.
- We identify elementary operations and measure running time in terms of these (e.g. comparisons, swaps for sorting).
- Every comparison-based sorting method must use $\Omega(n \log n)$ comparisons in the worst case for a file of size n.

Asymptotic notation

- Let $f, g : \mathbb{N} \to \mathbb{N}$. We say that $f \in O(g)$ if there is $C > 0$ such that $f(n) \leq Cg(n)$ for all sufficiently large n.
- $f \in \Omega(g) \Leftrightarrow g \in O(f)$.
- $f \in \Theta(g) \Leftrightarrow f \in O(g)$ and $f \in \Omega(g)$.
- In particular if $\lim_{n\to\infty} f(n)/g(n)$ exists and equals L, then

$$\begin{cases} f \in \Theta(g) & \text{if } 0 < L < \infty; \\ f \in \Omega(f) & \text{if } L = \infty; \\ f \in O(g) & \text{if } L = 0. \end{cases}$$

Our basic framework

- Let $\mathcal{A}$ be an algorithm for a given problem, and let $\mathcal{I}$ be the set of legal inputs. The size of input $\iota \in \mathcal{I}$ is denoted $|\iota|$; we let $\mathcal{I}_n$ be the set of inputs of size n. The running time of $\mathcal{A}$ on input ι is the number of elementary operations $T(\iota)$.
- Worst-case analysis studies $W(n) := \max_{\iota \in \mathcal{I}_n} T(\iota)$. Example: for sorting integers, $\mathcal{I}_n$ is the set of all sequences of integers of length n. For quicksort, $W(n) \in \Theta(n^2)$ but for mergesort, $W(n) \in \Theta(n \log n)$.
- Here we prefer to study the *distribution* of $T(\iota)$ over $\mathcal{I}_n$, since this is often more relevant in practice. This approach requires some probability model on the inputs, and then T is a random variable, whose mean, variance, etc, can be studied.
- We concentrate on systematic and powerful mathematical tools, avoiding special tricks.

Organizational matters

- Lecturer: Dr Mark Wilson; office 303.588; www.cs.auckland.ac.nz/~mcw. Office hours by appointment and whenever my door is open - email is preferred.
- Recommended reading:
  - Flajolet and Sedgewick, An Introduction to the Analysis of Algorithms (on reserve in library);
  - Wilf, generatingfunctionology (in library, also freely available online from my webpage);
  - Graham, Knuth, Patashnik, Concrete Mathematics (on reserve).
  - Lecture slides available online, but are continually being updated and corrected, so don't rely on them until the end of the course.
- One assignment worth 20% of course marks will be given.

Quicksort algorithm

- A recursive algorithm for sorting an array of n elements from a totally ordered set.
- If $n = 0$ or $n = 1$, do nothing. Otherwise, choose a pivot element x and partition the array so that if $y < x$ then y is to the left of x and if $z > x$ then z is to the right of x. Then call the algorithm recursively on the left and right parts.
- Note that we have not specified what happens if $x = y$, but we must do so in any implementation.
- The partitioning step requires at least $n - 1$ comparisons, plus some swaps.

Quicksort recurrence

- Let $C(F)$ be the number of comparisons done by quicksort on an n-element file F of *distinct* elements. Let F_1, F_2 be the subfiles consisting of elements less than, greater than the pivot. Then

$$C(F) = \begin{cases} 0 & \text{if } n = 0; \\ n - 1 + C(F_1) + C(F_2) & \text{if } n > 0. \end{cases}$$

Quicksort recurrence

- Let $C(F)$ be the number of comparisons done by quicksort on an n-element file F of *distinct* elements. Let F_1, F_2 be the subfiles consisting of elements less than, greater than the pivot. Then

$$C(F) = \begin{cases} 0 & \text{if } n = 0; \\ n - 1 + C(F_1) + C(F_2) & \text{if } n > 0. \end{cases}$$

- If the pivot is always the smallest element then F_1 is empty, so iterating this recursion gives $W(n) \in \Theta(n^2)$. Quicksort has a bad worst case.

Quicksort recurrence

- Let $C(F)$ be the number of comparisons done by quicksort on an n-element file F of *distinct* elements. Let F_1, F_2 be the subfiles consisting of elements less than, greater than the pivot. Then

$$C(F) = \begin{cases} 0 & \text{if } n = 0; \\ n - 1 + C(F_1) + C(F_2) & \text{if } n > 0. \end{cases}$$

- If the pivot is always the smallest element then F_1 is empty, so iterating this recursion gives $W(n) \in \Theta(n^2)$. Quicksort has a bad worst case.
- Choose an input file (permutation of $\{1, \ldots, n\}$) uniformly at random. After pivoting, the pivot is equally likely to be in any of the n positions, and each subfile is uniformly distributed, so

$$E[C_n] = n - 1 + \frac{1}{n} \sum_{p=1}^{n} (E[C_{p-1}] + E[C_{n-p}]).$$

Quicksort recurrence solution

- Let $a_n = E[C_n]$ so that $a_n = n - 1 + \frac{1}{n}\sum_{1\leq p\leq n}(a_{p-1} + a_{n-p})$. Collect common terms in the sums to obtain

$$a_n = n - 1 + \frac{2}{n}\sum_{j=0}^{n-1} a_j.$$

Quicksort recurrence solution

- Let $a_n = E[C_n]$ so that $a_n = n - 1 + \frac{1}{n}\sum_{1\leq p\leq n}(a_{p-1} + a_{n-p})$. Collect common terms in the sums to obtain

$$a_n = n - 1 + \frac{2}{n}\sum_{j=0}^{n-1} a_j.$$

- This full history recurrence can be solved by eliminating the history, (a general form of telescoping).

Quicksort recurrence solution

- Let $a_n = E[C_n]$ so that $a_n = n - 1 + \frac{1}{n}\sum_{1\le p\le n}(a_{p-1} + a_{n-p})$. Collect common terms in the sums to obtain

$$a_n = n - 1 + \frac{2}{n}\sum_{j=0}^{n-1} a_j.$$

- This full history recurrence can be solved by eliminating the history, (a general form of telescoping).
- The solution is $a_n = 2(n+1)H_n - 4n \approx 2n\log n$, where

$$H_n := \sum_{1\le j\le n} 1/j, \qquad \text{the } n\text{th harmonic number.}$$

Quicksort recurrence solution

- Let $a_n = E[C_n]$ so that $a_n = n - 1 + \frac{1}{n}\sum_{1\le p\le n}(a_{p-1} + a_{n-p})$. Collect common terms in the sums to obtain

$$a_n = n - 1 + \frac{2}{n}\sum_{j=0}^{n-1} a_j.$$

- This full history recurrence can be solved by eliminating the history, (a general form of telescoping).
- The solution is $a_n = 2(n+1)H_n - 4n \approx 2n\log n$, where

$$H_n := \sum_{1\le j\le n} 1/j, \qquad \text{the } n\text{th harmonic number.}$$

- Quicksort is of optimal order on average.

Quicksort recurrence solution

- Let $a_n = E[C_n]$ so that $a_n = n - 1 + \frac{1}{n}\sum_{1\le p\le n}(a_{p-1} + a_{n-p})$. Collect common terms in the sums to obtain

$$a_n = n - 1 + \frac{2}{n}\sum_{j=0}^{n-1} a_j.$$

- This full history recurrence can be solved by eliminating the history, (a general form of telescoping).
- The solution is $a_n = 2(n+1)H_n - 4n \approx 2n\log n$, where

$$H_n := \sum_{1\le j\le n} 1/j, \qquad \text{the } n\text{th harmonic number.}$$

- Quicksort is of optimal order on average.
- What about the variance?

Quicksort recurrence solution details

- We have for $n > 1$

$$na_n - (n-1)a_{n-1} = n(n-1) + 2\sum_{0 \le j < n} a_j$$
$$- (n-1)(n-2) - 2\sum_{0 \le j < n-1} a_j$$

Quicksort recurrence solution details

- We have for $n > 1$

$$\begin{aligned} na_n - (n-1)a_{n-1} &= n(n-1) + 2\sum_{0\le j<n} a_j \\ &\quad - (n-1)(n-2) - 2\sum_{0\le j<n-1} a_j \\ &= 2(n-1) + 2a_{n-1} \end{aligned}$$

- Thus $na_n = 2(n-1) + (n+1)a_{n-1}$, so that

$$\frac{a_n}{n+1} = \frac{2(n-1)}{n(n+1)} + \frac{a_{n-1}}{n} = \frac{a_{n-1}}{n} + 2(n-1).$$

Quicksort recurrence solution details

- We have for $n > 1$

$$\begin{aligned} na_n - (n-1)a_{n-1} &= n(n-1) + 2\sum_{0\le j<n} a_j \\ &\quad - (n-1)(n-2) - 2\sum_{0\le j<n-1} a_j \\ &= 2(n-1) + 2a_{n-1} \end{aligned}$$

- Thus $na_n = 2(n-1) + (n+1)a_{n-1}$, so that

$$\frac{a_n}{n+1} = \frac{2(n-1)}{n(n+1)} + \frac{a_{n-1}}{n} = \frac{a_{n-1}}{n} + 2(n-1).$$

- We can iterate to obtain

$$\frac{a_n}{n+1} = \frac{a_1}{2} + \sum_{j=2}^{n} \frac{2}{j+1} + \frac{1}{j} - \frac{1}{j+1}.$$

Issues arising from our analysis

- How do we analyse suggested improvements, such as using a cutoff for small files, or median-of-3 pivoting, or different partitioning method?

We answer points 1–4 in this course.

Issues arising from our analysis

- How do we analyse suggested improvements, such as using a cutoff for small files, or median-of-3 pivoting, or different partitioning method?
- The method we used for solving the recurrences was somewhat specialized. How to do it more generally?

We answer points 1–4 in this course.

Issues arising from our analysis

- How do we analyse suggested improvements, such as using a cutoff for small files, or median-of-3 pivoting, or different partitioning method?
- The method we used for solving the recurrences was somewhat specialized. How to do it more generally?
- How do we get more information about the full distribution of the number of comparisons?

We answer points 1–4 in this course.

Issues arising from our analysis

- How do we analyse suggested improvements, such as using a cutoff for small files, or median-of-3 pivoting, or different partitioning method?
- The method we used for solving the recurrences was somewhat specialized. How to do it more generally?
- How do we get more information about the full distribution of the number of comparisons?
- How to derive asymptotics for the sums occurring?

We answer points 1–4 in this course.

Issues arising from our analysis

- How do we analyse suggested improvements, such as using a cutoff for small files, or median-of-3 pivoting, or different partitioning method?
- The method we used for solving the recurrences was somewhat specialized. How to do it more generally?
- How do we get more information about the full distribution of the number of comparisons?
- How to derive asymptotics for the sums occurring?
- The above analysis relies on the fact that the subfiles are themselves uniformly distributed. What to do if this is not the case?

We answer points 1–4 in this course.

Issues arising from our analysis

- How do we analyse suggested improvements, such as using a cutoff for small files, or median-of-3 pivoting, or different partitioning method?
- The method we used for solving the recurrences was somewhat specialized. How to do it more generally?
- How do we get more information about the full distribution of the number of comparisons?
- How to derive asymptotics for the sums occurring?
- The above analysis relies on the fact that the subfiles are themselves uniformly distributed. What to do if this is not the case?
- How to analyse input where keys can be equal?

We answer points 1–4 in this course.

Ordinary generating functions — OGFs

- The OGF associated to a sequence $a_0, a_1, \ldots,$ is the formal power series $F(z) = \sum_n a_n z^n$. Examples: $1, 1, 1, \ldots,$ has OGF $1 + z + z^2 + \cdots = 1/(1-z)$. See handout for more examples.

Ordinary generating functions — OGFs

- The OGF associated to a sequence $a_0, a_1, \ldots$, is the formal power series $F(z) = \sum_n a_n z^n$. Examples: $1, 1, 1, \ldots$, has OGF $1 + z + z^2 + \cdots = 1/(1-z)$. See handout for more examples.
- Basic operations on sequences (sum, convolution, ...) correspond to those on OGFs (sum, product, ...). See handout for more operations.

Ordinary generating functions — OGFs

- The OGF associated to a sequence $a_0, a_1, \ldots$, is the formal power series $F(z) = \sum_n a_n z^n$. Examples: $1, 1, 1, \ldots$, has OGF $1 + z + z^2 + \cdots = 1/(1-z)$. See handout for more examples.
- Basic operations on sequences (sum, convolution, ...) correspond to those on OGFs (sum, product, ...). See handout for more operations.
- The equality $\sum_n z^n = 1/(1-z)$ is purely formal at this stage but also makes sense for $|z| < 1$. So far OGFs are just a convenient short way to describe sequences, but soon they will be a powerful machine.

Ordinary generating functions — OGFs

- The OGF associated to a sequence $a_0, a_1, \ldots$, is the formal power series $F(z) = \sum_n a_n z^n$. Examples: $1, 1, 1, \ldots$, has OGF $1 + z + z^2 + \cdots = 1/(1-z)$. See handout for more examples.
- Basic operations on sequences (sum, convolution, ...) correspond to those on OGFs (sum, product, ...). See handout for more operations.
- The equality $\sum_n z^n = 1/(1-z)$ is purely formal at this stage but also makes sense for $|z| < 1$. So far OGFs are just a convenient short way to describe sequences, but soon they will be a powerful machine.
- Given an OGF, how to extract its sequence if not available from above list? Taylor series definition always works, but is usually not computationally useful. We mainly use table lookup here.

Summary - power series

- Basic principle: if an identity between power series is true at the level of analytic functions, then it is true for formal series, provided all operations concerned are formally valid (computation of each coefficient can be carried out in B).

Summary - power series

- Basic principle: if an identity between power series is true at the level of analytic functions, then it is true for formal series, provided all operations concerned are formally valid (computation of each coefficient can be carried out in B).
- Thus we may freely use all our algebra and calculus knowledge, with appropriate caution about composition.

Summary - power series

- Basic principle: if an identity between power series is true at the level of analytic functions, then it is true for formal series, provided all operations concerned are formally valid (computation of each coefficient can be carried out in B).
- Thus we may freely use all our algebra and calculus knowledge, with appropriate caution about composition.
- An example of what is not allowed formally: $\sum_n (z+1)^n/n! = e \sum_n z^n/n!$. This is true at the function level for every $z \in \mathbb{C}$ and says $\exp(z+1) = \exp(1)\exp(z)$, but the constant term of the left side can't be computed in B.

Formal power series

- A sequence is a function $f : \mathbb{N} \to \mathbb{C}$. We sometimes write $(f(0), f(1), \ldots,)$ instead of just f. The special sequences $1 := (1, 0, 0, \ldots,)$ and $z := (0, 1, 0, 0 \ldots)$ will be useful.

Formal power series

- A sequence is a function $f : \mathbb{N} \to \mathbb{C}$. We sometimes write $(f(0), f(1), \ldots,)$ instead of just f. The special sequences $1 := (1, 0, 0, \ldots,)$ and $z := (0, 1, 0, 0 \ldots)$ will be useful.
- The set of all sequences we call A. We define operations $+$ and $\cdot$ on A as follows.

With these operations and the obvious multiplication by complex numbers, A becomes a commutative associative algebra. It contains a subalgebra B (consisting of all sequences with finitely many nonzero terms) isomorphic to the algebra of polynomials in one indeterminate.

Formal power series

- A sequence is a function $f : \mathbb{N} \to \mathbb{C}$. We sometimes write $(f(0), f(1), \ldots,)$ instead of just f. The special sequences $1 := (1, 0, 0, \ldots,)$ and $z := (0, 1, 0, 0 \ldots)$ will be useful.
- The set of all sequences we call A. We define operations $+$ and $\cdot$ on A as follows.
  - The sum $f + g$ of sequences is the sequence h such that $h(n) = f(n) + g(n)$ for each $n \in \mathbb{N}$.

With these operations and the obvious multiplication by complex numbers, A becomes a commutative associative algebra. It contains a subalgebra B (consisting of all sequences with finitely many nonzero terms) isomorphic to the algebra of polynomials in one indeterminate.

Formal power series

- A sequence is a function $f : \mathbb{N} \to \mathbb{C}$. We sometimes write $(f(0), f(1), \dots,)$ instead of just f. The special sequences $1 := (1, 0, 0, \dots,)$ and $z := (0, 1, 0, 0 \dots)$ will be useful.
- The set of all sequences we call A. We define operations $+$ and $\cdot$ on A as follows.
  - The sum $f + g$ of sequences is the sequence h such that $h(n) = f(n) + g(n)$ for each $n \in \mathbb{N}$.
  - The product $f \cdot g$ is the sequence h such that for each n, $h(n) = \sum_{k=0}^{n} f(k)g(n-k)$.

  With these operations and the obvious multiplication by complex numbers, A becomes a commutative associative algebra. It contains a subalgebra B (consisting of all sequences with finitely many nonzero terms) isomorphic to the algebra of polynomials in one indeterminate.

Operations in the power series algebra

- We can define the derivative of f as the sequence h such that $h(n) = (n+1)f(n+1)$.

All operations take place in B; that is, computing $h(n)$ always involves only finite algebra.

Operations in the power series algebra

- We can define the derivative of f as the sequence h such that $h(n) = (n+1)f(n+1)$.
- The antiderivative of f is the sequence h such that $h(n) = f(n-1)/n$ for $n > 0$, and $h(0) = 0$.

All operations take place in B; that is, computing $h(n)$ always involves only finite algebra.

Operations in the power series algebra

- We can define the derivative of f as the sequence h such that $h(n) = (n+1)f(n+1)$.
- The antiderivative of f is the sequence h such that $h(n) = f(n-1)/n$ for $n > 0$, and $h(0) = 0$.
- The composition of f and g is the sequence h given by $h(n) = \sum_{k=0}^{n} f(k)g^k$. This is only valid when $g(0) = 0$.

All operations take place in B; that is, computing $h(n)$ always involves only finite algebra.

Operations in the power series algebra

- We can define the derivative of f as the sequence h such that $h(n) = (n+1)f(n+1)$.
- The antiderivative of f is the sequence h such that $h(n) = f(n-1)/n$ for $n > 0$, and $h(0) = 0$.
- The composition of f and g is the sequence h given by $h(n) = \sum_{k=0}^{n} f(k)g^k$. This is only valid when $g(0) = 0$.
- The inverse of f is the series h such that $f \cdot h = 1$; this exists if and only if $f(0) \neq 0$.

All operations take place in B; that is, computing $h(n)$ always involves only finite algebra.

Analytic functions and power series

Some facts from a course in complex analysis (not proved here):

- For each $f \in A$ there is a largest $R \leq \infty$, such that for $|z| < R$, the infinite series $\sum_n f(n)z^n$ converges absolutely and uniformly to a limit $F(z)$.

Analytic functions and power series

Some facts from a course in complex analysis (not proved here):

- For each $f \in A$ there is a largest $R \leq \infty$, such that for $|z| < R$, the infinite series $\sum_n f(n)z^n$ converges absolutely and uniformly to a limit $F(z)$.
- If $R > 0$, then F is analytic (has derivatives of all orders) for $|z| < R$, and can be integrated and differentiated term-by-term.

Analytic functions and power series

Some facts from a course in complex analysis (not proved here):

- For each $f \in A$ there is a largest $R \leq \infty$, such that for $|z| < R$, the infinite series $\sum_n f(n)z^n$ converges absolutely and uniformly to a limit $F(z)$.
- If $R > 0$, then F is analytic (has derivatives of all orders) for $|z| < R$, and can be integrated and differentiated term-by-term.
- Conversely, if $R > 0$ and F is an analytic function for $|z| < R$, then there is a unique sequence f so that $F(z) = \sum_n f(n)z^n$.

Analytic functions and power series

Some facts from a course in complex analysis (not proved here):

- For each $f \in A$ there is a largest $R \leq \infty$, such that for $|z| < R$, the infinite series $\sum_n f(n)z^n$ converges absolutely and uniformly to a limit $F(z)$.
- If $R > 0$, then F is analytic (has derivatives of all orders) for $|z| < R$, and can be integrated and differentiated term-by-term.
- Conversely, if $R > 0$ and F is an analytic function for $|z| < R$, then there is a unique sequence f so that $F(z) = \sum_n f(n)z^n$.
- Here f is essentially the sequence of Taylor coefficients of F at 0,

$$f(n) = \frac{1}{n!}\left(\frac{d}{dz}\right)^n F(z)_{|z=0}$$

and can also be computed by Cauchy's integral formula

$$f(n) = \frac{1}{2\pi i}\int_C \frac{F(z)}{z^{n+1}}\,dz.$$

Exponential generating functions — EGFs

- The EGF associated to a sequence $a_0, a_1, \ldots,$ is the formal power series $F(z) = \sum_n a_n z^n/n!$. Examples: $1, 1, 1, \ldots,$ has EGF $1 + z + z^2/2! + \cdots = \exp(z)$. See handout.

Exponential generating functions — EGFs

- The EGF associated to a sequence $a_0, a_1, \ldots,$ is the formal power series $F(z) = \sum_n a_n z^n/n!$. Examples: $1, 1, 1, \ldots,$ has EGF $1 + z + z^2/2! + \cdots = \exp(z)$. See handout.
- The EGF of sequence $\{a_n\}$ is the OGF of $\{a_n/n!\}$.

Exponential generating functions — EGFs

- The EGF associated to a sequence $a_0, a_1, \ldots,$ is the formal power series $F(z) = \sum_n a_n z^n/n!$. Examples: $1, 1, 1, \ldots,$ has EGF $1 + z + z^2/2! + \cdots = \exp(z)$. See handout.
- The EGF of sequence $\{a_n\}$ is the OGF of $\{a_n/n!\}$.
- EGFs are often used for labelled constructions and for permutations (perhaps more later).

Exponential generating functions — EGFs

- The EGF associated to a sequence $a_0, a_1, \ldots,$ is the formal power series $F(z) = \sum_n a_n z^n/n!$. Examples: $1, 1, 1, \ldots,$ has EGF $1 + z + z^2/2! + \cdots = \exp(z)$. See handout.
- The EGF of sequence $\{a_n\}$ is the OGF of $\{a_n/n!\}$.
- EGFs are often used for labelled constructions and for permutations (perhaps more later).
- Basic operations on sequences (sum, convolution, ...) correspond to those on EGFs (sum, product, ...). See handout.

Exponential generating functions — EGFs

- The EGF associated to a sequence $a_0, a_1, \ldots,$ is the formal power series $F(z) = \sum_n a_n z^n/n!$. Examples: $1, 1, 1, \ldots,$ has EGF $1 + z + z^2/2! + \cdots = \exp(z)$. See handout.
- The EGF of sequence $\{a_n\}$ is the OGF of $\{a_n/n!\}$.
- EGFs are often used for labelled constructions and for permutations (perhaps more later).
- Basic operations on sequences (sum, convolution, ...) correspond to those on EGFs (sum, product, ...). See handout.
- Sometimes OGFs are better for computation, sometimes EGFs.

Probability generating functions — PGFs

- The PGF associated to a random variable X taking values in $\mathbb{N}$ is $G(z) = \sum_n \Pr(X = n) z^n$.

Probability generating functions — PGFs

- The PGF associated to a random variable X taking values in $\mathbb{N}$ is $G(z) = \sum_n \Pr(X = n) z^n$.
- The mean of X is just $G'(1)$ and the variance is $G''(1) + G'(1) - G'(1)^2$.

Probability generating functions — PGFs

- The PGF associated to a random variable X taking values in $\mathbb{N}$ is $G(z) = \sum_n \Pr(X = n) z^n$.
- The mean of X is just $G'(1)$ and the variance is $G''(1) + G'(1) - G'(1)^2$.
- The PGF of the sum of independent RVs X and Y is the product of the individual PGFs.

Probability generating functions — PGFs

- The PGF associated to a random variable X taking values in $\mathbb{N}$ is $G(z) = \sum_n \Pr(X = n) z^n$.
- The mean of X is just $G'(1)$ and the variance is $G''(1) + G'(1) - G'(1)^2$.
- The PGF of the sum of independent RVs X and Y is the product of the individual PGFs.
- The PGF of the sequence $\Pr(X > n)$ of tail probabilities is $(1 - G(z))/(1 - z)$.

Multivariate generating functions — MGFs

Occur naturally in many contexts. An important one for probability is as follows. Let a_{nk} be the number of objects of some type with size n and another parameter χ equal to k. Let $F(z,u) = \sum a_{nk} z^n u^k$, the bivariate GF. Then

- $c_n := [z^n]F(z,1) = \sum_k a_{nk}$ is the number of objects of size n;
- $\mu_n := (1/c_n)[z^n]F_u(z,1) = (1/c_n)\sum_k k a_{nk}$ is the mean of χ on a uniformly chosen object of size n;
- $\sigma_n^2 := (1/c_n)[z^n]F_{uu}(z,1) + \mu_n - \mu_n^2$ is the variance of χ on a uniformly chosen object of size n.

GFs solve recurrences

- Main idea: recurrence gives an equation involving the GF. Try to solve this, then extract the coefficients.

GFs solve recurrences

- Main idea: recurrence gives an equation involving the GF. Try to solve this, then extract the coefficients.
- Example (Fibonacci): $a_0 = 0, a_1 = 1; a_n = a_{n-1} + a_{n-2}$ for $n \geq 2$. Multiply each side by z^n, then sum on n. Let $F(z) = \sum_n a_n z^n$. Then we get $F(z) - z = zF(z) + z^2 F(z)$. This gives $F(z) = z/(1 - z - z^2)$.

GFs solve recurrences

- Main idea: recurrence gives an equation involving the GF. Try to solve this, then extract the coefficients.
- Example (Fibonacci): $a_0 = 0, a_1 = 1; a_n = a_{n-1} + a_{n-2}$ for $n \geq 2$. Multiply each side by z^n, then sum on n. Let $F(z) = \sum_n a_n z^n$. Then we get $F(z) - z = zF(z) + z^2 F(z)$. This gives $F(z) = z/(1 - z - z^2)$.
- To convert back to a sequence, we can use partial fractions. Get $F(z) = A/(1 - \phi z) + B/(1 + \phi^{-1} z)$ where $\phi = 1.618\cdots$, the golden ratio, $A = 1/\sqrt{5}, B = -1/\sqrt{5}$.

GFs solve recurrences

- Main idea: recurrence gives an equation involving the GF. Try to solve this, then extract the coefficients.
- Example (Fibonacci): $a_0 = 0, a_1 = 1; a_n = a_{n-1} + a_{n-2}$ for $n \geq 2$. Multiply each side by z^n, then sum on n. Let $F(z) = \sum_n a_n z^n$. Then we get $F(z) - z = zF(z) + z^2 F(z)$. This gives $F(z) = z/(1 - z - z^2)$.
- To convert back to a sequence, we can use partial fractions. Get $F(z) = A/(1 - \phi z) + B/(1 + \phi^{-1} z)$ where $\phi = 1.618\cdots$, the golden ratio, $A = 1/\sqrt{5}, B = -1/\sqrt{5}$.
- Thus $a_n = (\phi^n + (-1)^n \phi^{-n})/\sqrt{5} \in \Theta(\phi^n)$.

GFs solve recurrences

- Main idea: recurrence gives an equation involving the GF. Try to solve this, then extract the coefficients.
- Example (Fibonacci): $a_0 = 0, a_1 = 1; a_n = a_{n-1} + a_{n-2}$ for $n \geq 2$. Multiply each side by z^n, then sum on n. Let $F(z) = \sum_n a_n z^n$. Then we get $F(z) - z = zF(z) + z^2 F(z)$. This gives $F(z) = z/(1 - z - z^2)$.
- To convert back to a sequence, we can use partial fractions. Get $F(z) = A/(1 - \phi z) + B/(1 + \phi^{-1} z)$ where $\phi = 1.618\cdots$, the golden ratio, $A = 1/\sqrt{5}, B = -1/\sqrt{5}$.
- Thus $a_n = (\phi^n + (-1)^n \phi^{-n})/\sqrt{5} \in \Theta(\phi^n)$.
- Note: this can be done automatically by a computer algebra system!

Quicksort recurrence with GFs

- From $na_n = n(n-1) + 2\sum_{j<n} a_j$, $a_0 = 0$, we obtain $zF'(z) = 2z^2/(1-z)^3 + 2zF(z)/(1-z)$ via the dictionary.

Quicksort recurrence with GFs

- From $na_n = n(n-1) + 2\sum_{j<n} a_j$, $a_0 = 0$, we obtain $zF'(z) = 2z^2/(1-z)^3 + 2zF(z)/(1-z)$ via the dictionary.
- This is a standard first order linear inhomogeneous differential equation that can be solved (how?) to obtain

$$F(z) = \frac{2}{(1-z)^2}\left(\log\frac{1}{1-z} - z\right).$$

Thus by lookup we have $a_n = 2(n+1)H_n - 4n$.

GFs also yield recurrences

- Example: for fixed $r \geq 1$, consider

$$F(z) = (1-z)/(1-2z+z^{r+1}) = \sum_n a_n z^n.$$

Thus $(\sum_n a_n z^n)(1-2z+z^{r+1}) = 1-z$. Compare coefficients to see that $a_0 = 1, a_1 - 2a_0 = -1$, and

$$a_n - 2a_{n-1} + a_{n-r-1} = 0 \qquad \text{for } n \geq 2.$$

This allows linear time computation of a_n.

GFs also yield recurrences

- Example: for fixed $r \geq 1$, consider

$$F(z) = (1-z)/(1-2z+z^{r+1}) = \sum_n a_n z^n.$$

Thus $(\sum_n a_n z^n)(1-2z+z^{r+1}) = 1-z$. Compare coefficients to see that $a_0 = 1, a_1 - 2a_0 = -1$, and

$$a_n - 2a_{n-1} + a_{n-r-1} = 0 \qquad \text{for } n \geq 2.$$

This allows linear time computation of a_n.

- Every rational OGF gives a linear constant coefficient recurrence in this way, and vice versa.

GFs also yield recurrences

- Example: for fixed $r \geq 1$, consider

$$F(z) = (1-z)/(1-2z+z^{r+1}) = \sum_n a_n z^n.$$

Thus $(\sum_n a_n z^n)(1-2z+z^{r+1}) = 1-z$. Compare coefficients to see that $a_0 = 1, a_1 - 2a_0 = -1$, and

$$a_n - 2a_{n-1} + a_{n-r-1} = 0 \qquad \text{for } n \geq 2.$$

This allows linear time computation of a_n.

- Every rational OGF gives a linear constant coefficient recurrence in this way, and vice versa.
- In the same way, linear recurrences with polynomial coefficients correspond to linear differential equations for the OGF.

GFs sometimes yield better recurrences

- The counting GF of binary trees by internal nodes satisfies $T(z) = 1 + zT(z)^2$ (later). This is equivalent to the quadratic recurrence $a_0 = 1$, $a_n = \sum_{k<n} a_k a_{n-1-k}$ for the number a_n of binary trees with n nodes.

GFs sometimes yield better recurrences

- The counting GF of binary trees by internal nodes satisfies $T(z) = 1 + zT(z)^2$ (later). This is equivalent to the quadratic recurrence $a_0 = 1$, $a_n = \sum_{k<n} a_k a_{n-1-k}$ for the number a_n of binary trees with n nodes.
- There is an algorithm (Comtet 1964) which finds a linear differential equation with polynomial coefficients for each algebraic GF.

GFs sometimes yield better recurrences

- The counting GF of binary trees by internal nodes satisfies $T(z) = 1 + zT(z)^2$ (later). This is equivalent to the quadratic recurrence $a_0 = 1$, $a_n = \sum_{k<n} a_k a_{n-1-k}$ for the number a_n of binary trees with n nodes.
- There is an algorithm (Comtet 1964) which finds a linear differential equation with polynomial coefficients for each algebraic GF.
- In this case the answer turns out to be

$$(4z^2 - z)T'(z) + (2z - 1)T(z) + 1 = 0$$

which is equivalent to the recurrence

$$(n+1)a_n = (4n-2)a_{n-1} \qquad a_0 = 1.$$

This allows for much faster computation and makes it plain that a_n involves a quotient of factorials (in fact $a_n = \frac{1}{n+1}\binom{2n}{n}$, the nth Catalan number).

Comtet's algorithm outline

- Suppose $P(z, F(z)) = 0$ where $P(z, y) \in \mathbb{C}[z, y]$ is irreducible. Differentiate and solve for F' to obtain

$$F'(z) = \frac{A(z, y)}{B(z, y)} \qquad \text{for relatively prime polynomials } A, B \in \mathbb{C}[z, y].$$

Note that B and P are relatively prime.

Comtet's algorithm outline

- Suppose $P(z, F(z)) = 0$ where $P(z, y) \in \mathbb{C}[z, y]$ is irreducible. Differentiate and solve for F' to obtain

$$F'(z) = \frac{A(z, y)}{B(z, y)} \qquad \text{for relatively prime polynomials } A, B \in \mathbb{C}[z, y].$$

  Note that B and P are relatively prime.

- The extended Euclidean algorithm yields polynomials $u(z, y), v(z, y), g(z)$ such that $uB + vP = g$. Define $C = Au \mod P$ to get $F'(z) = C(y, z)/g(z)$. Repeat as required.

Comtet's algorithm outline

- Suppose $P(z, F(z)) = 0$ where $P(z, y) \in \mathbb{C}[z, y]$ is irreducible. Differentiate and solve for F' to obtain

$$F'(z) = \frac{A(z, y)}{B(z, y)} \qquad \text{for relatively prime polynomials } A, B \in \mathbb{C}[z, y].$$

  Note that B and P are relatively prime.
- The extended Euclidean algorithm yields polynomials $u(z, y), v(z, y), g(z)$ such that $uB + vP = g$. Define $C = Au \mod P$ to get $F'(z) = C(y, z)/g(z)$. Repeat as required.
- In above binary tree example, $P = zy^2 - y + 1, B = 1 - 2zy, A = y^2, u = (2zy-1)/(4z-1), C = (2zy-y-z)/(z-4z^2), g = 1/(4z-1)$. Algorithm terminates in one step.

OGFs and counting I

- Can often set up a recursion and solve as above. Example: how many binary trees T_n with n (internal) nodes? Binary tree is a recursive object: either a single external node, or an internal node connected to an ordered pair of binary trees. Thus $T_0 = 1$ and for $n > 0$,

$$T_n = \sum_{1 \le k \le n} T_{k-1} T_{n-k} = \sum_{0 \le k \le n-1} T_k T_{n-1-k}.$$

OGFs and counting I

- Can often set up a recursion and solve as above. Example: how many binary trees T_n with n (internal) nodes? Binary tree is a recursive object: either a single external node, or an internal node connected to an ordered pair of binary trees. Thus $T_0 = 1$ and for $n > 0$,

$$T_n = \sum_{1 \le k \le n} T_{k-1} T_{n-k} = \sum_{0 \le k \le n-1} T_k T_{n-1-k}.$$

- Hence OGF satisfies $T(z) = zT(z)^2 + 1$. Thus $T(z) = (1 - \sqrt{1-4z})/2z$ (how to choose square root?).

OGFs and counting I

- Can often set up a recursion and solve as above. Example: how many binary trees T_n with n (internal) nodes? Binary tree is a recursive object: either a single external node, or an internal node connected to an ordered pair of binary trees. Thus $T_0 = 1$ and for $n > 0$,

$$T_n = \sum_{1 \le k \le n} T_{k-1} T_{n-k} = \sum_{0 \le k \le n-1} T_k T_{n-1-k}.$$

- Hence OGF satisfies $T(z) = zT(z)^2 + 1$. Thus $T(z) = (1 - \sqrt{1-4z})/2z$ (how to choose square root?).
- Can extract coefficients using binomial theorem: get

$$T_n = \frac{1}{n+1}\binom{2n}{n} \quad \text{(Catalan number)}.$$

OGFs and counting I

- Can often set up a recursion and solve as above. Example: how many binary trees T_n with n (internal) nodes? Binary tree is a recursive object: either a single external node, or an internal node connected to an ordered pair of binary trees. Thus $T_0 = 1$ and for $n > 0$,

$$T_n = \sum_{1 \le k \le n} T_{k-1} T_{n-k} = \sum_{0 \le k \le n-1} T_k T_{n-1-k}.$$

- Hence OGF satisfies $T(z) = zT(z)^2 + 1$. Thus $T(z) = (1 - \sqrt{1-4z})/2z$ (how to choose square root?).
- Can extract coefficients using binomial theorem: get

$$T_n = \frac{1}{n+1}\binom{2n}{n} \quad \text{(Catalan number)}.$$

- Asymptotics can be derived (later) and we will get $T_n \sim 4^n/\sqrt{\pi n^3}$.

OGFs and counting II

- A nicer way to get the OGF is as follows. Let $\mathcal{T}$ be the set of all binary trees, $|t|$ the size of tree t, $\mathcal{T}_n = \{t \in \mathcal{T} : |t| = n\}$. Then

$$\begin{aligned} T(z) &= \sum_n \sum_{t \in \mathcal{T}_n} z^n = \sum_{t \in \mathcal{T}} z^{|t|} = 1 + \sum_{t \in \mathcal{T} \setminus \mathcal{T}_0} z^{|t_l|+|t_r|+1} \\ &= 1 + \sum_{t_l \in \mathcal{T}} \sum_{t_r \in \mathcal{T}} z^{|t_l|+|t_r|+1} = 1 + \sum_{t_l \in \mathcal{T}} z^{|t_l|} \sum_{t_r \in \mathcal{T}} z^{|t_r|} \\ &= 1 + zT(z)^2 \end{aligned}$$

where t_r, t_l are the right and left subtrees of t.

OGFs and counting II

- A nicer way to get the OGF is as follows. Let $\mathcal{T}$ be the set of all binary trees, $|t|$ the size of tree t, $\mathcal{T}_n = \{t \in \mathcal{T} : |t| = n\}$. Then

$$\begin{aligned} T(z) &= \sum_n \sum_{t \in \mathcal{T}_n} z^n = \sum_{t \in \mathcal{T}} z^{|t|} = 1 + \sum_{t \in \mathcal{T} \setminus \mathcal{T}_0} z^{|t_l|+|t_r|+1} \\ &= 1 + \sum_{t_l \in \mathcal{T}} \sum_{t_r \in \mathcal{T}} z^{|t_l|+|t_r|+1} = 1 + \sum_{t_l \in \mathcal{T}} z^{|t_l|} \sum_{t_r \in \mathcal{T}} z^{|t_r|} \\ &= 1 + zT(z)^2 \end{aligned}$$

  where t_r, t_l are the right and left subtrees of t.

- Study this example carefully - it leads to the symbolic method for enumeration.

The symbolic method I

- Avoids explicit recurrences, goes straight from recursive description of structure to the counting GF. A great time-saver, and also more amenable to computer algebra implementation.

The symbolic method I

- Avoids explicit recurrences, goes straight from recursive description of structure to the counting GF. A great time-saver, and also more amenable to computer algebra implementation.
- A combinatorial class is a set X which is the disjoint union of finite sets X_n. The size $|x|$ of an element x is the value of n for which $x \in X_n$.

The symbolic method I

- Avoids explicit recurrences, goes straight from recursive description of structure to the counting GF. A great time-saver, and also more amenable to computer algebra implementation.
- A combinatorial class is a set X which is the disjoint union of finite sets X_n. The size $|x|$ of an element x is the value of n for which $x \in X_n$.
- Let $\mathcal{A}$ be a combinatorial class, with $|\mathcal{A}_n| = a_n$. The counting OGF for $\mathcal{A}$ is $A(z) = \sum_n a_n z^n = \sum_{a \in \mathcal{A}} z^{|a|}$.

The symbolic method I

- Avoids explicit recurrences, goes straight from recursive description of structure to the counting GF. A great time-saver, and also more amenable to computer algebra implementation.
- A combinatorial class is a set X which is the disjoint union of finite sets X_n. The size $|x|$ of an element x is the value of n for which $x \in X_n$.
- Let $\mathcal{A}$ be a combinatorial class, with $|\mathcal{A}_n| = a_n$. The counting OGF for $\mathcal{A}$ is $A(z) = \sum_n a_n z^n = \sum_{a \in \mathcal{A}} z^{|a|}$.
- Set operations such as disjoint union, cartesian product, sequence correspond to GF operations sum, product, quasi-inverse.

The symbolic method I

- Avoids explicit recurrences, goes straight from recursive description of structure to the counting GF. A great time-saver, and also more amenable to computer algebra implementation.
- A combinatorial class is a set X which is the disjoint union of finite sets X_n. The size $|x|$ of an element x is the value of n for which $x \in X_n$.
- Let $\mathcal{A}$ be a combinatorial class, with $|\mathcal{A}_n| = a_n$. The counting OGF for $\mathcal{A}$ is $A(z) = \sum_n a_n z^n = \sum_{a \in \mathcal{A}} z^{|a|}$.
- Set operations such as disjoint union, cartesian product, sequence correspond to GF operations sum, product, quasi-inverse.
- Main examples: plane trees, compositions, regular languages.

The symbolic method II

- Let $\mathcal{A}, \mathcal{B}$ be combinatorial classes with counting OGFs $A(z), B(z)$. The size of an ordered pair of objects (α, β) is defined to be $|\alpha| + |\beta|$.

The symbolic method II

- Let $\mathcal{A}, \mathcal{B}$ be combinatorial classes with counting OGFs $A(z), B(z)$. The size of an ordered pair of objects (α, β) is defined to be $|\alpha| + |\beta|$.
- The counting OGF of $\mathcal{A} \times \mathcal{B}$ is then

$$\sum_{\gamma \in \mathcal{A} \times \mathcal{B}} z^{|\gamma|} = \sum_{\alpha \in \mathcal{A}} \sum_{\beta \in \mathcal{B}} z^{|\alpha|+|\beta|} = A(z)B(z).$$

The symbolic method II

- Let $\mathcal{A}, \mathcal{B}$ be combinatorial classes with counting OGFs $A(z), B(z)$. The size of an ordered pair of objects (α, β) is defined to be $|\alpha| + |\beta|$.
- The counting OGF of $\mathcal{A} \times \mathcal{B}$ is then

$$\sum_{\gamma \in \mathcal{A} \times \mathcal{B}} z^{|\gamma|} = \sum_{\alpha \in \mathcal{A}} \sum_{\beta \in \mathcal{B}} z^{|\alpha|+|\beta|} = A(z)B(z).$$

- Also the counting OGF for $\mathcal{A} \cup \mathcal{B}$ is $A(z) + B(z)$ if the classes are disjoint. Thus the OGF for the set of sequences of elements of $\mathcal{A}$ is $1 + A(z) + A(z)^2 + \cdots = (1 - A(z))^{-1}$.

The symbolic method II

- Let $\mathcal{A}, \mathcal{B}$ be combinatorial classes with counting OGFs $A(z), B(z)$. The size of an ordered pair of objects (α, β) is defined to be $|\alpha| + |\beta|$.
- The counting OGF of $\mathcal{A} \times \mathcal{B}$ is then

$$\sum_{\gamma \in \mathcal{A} \times \mathcal{B}} z^{|\gamma|} = \sum_{\alpha \in \mathcal{A}} \sum_{\beta \in \mathcal{B}} z^{|\alpha|+|\beta|} = A(z)B(z).$$

- Also the counting OGF for $\mathcal{A} \cup \mathcal{B}$ is $A(z) + B(z)$ if the classes are disjoint. Thus the OGF for the set of sequences of elements of $\mathcal{A}$ is $1 + A(z) + A(z)^2 + \cdots = (1 - A(z))^{-1}$.
- If a combinatorial class is constructed from atoms using only disjoint union, product and sequence constructions, then its counting OGF is rational.

The symbolic method for EGFs

- EGFS are often used for labelled constructions. A labelled combinatorial object is one where each atom carries a positive integer label and all labels are distinct. The labelling is proper if the label set is $\{1, \ldots, n\}$.

The symbolic method for EGFs

- EGFS are often used for labelled constructions. A labelled combinatorial object is one where each atom carries a positive integer label and all labels are distinct. The labelling is proper if the label set is $\{1, \ldots, n\}$.
- Example: a permutation is a properly labelled sequence of atoms.

The symbolic method for EGFs

- EGFS are often used for labelled constructions. A labelled combinatorial object is one where each atom carries a positive integer label and all labels are distinct. The labelling is proper if the label set is $\{1, \ldots, n\}$.
- Example: a permutation is a properly labelled sequence of atoms.
- When forming the sum or product of labelled classes, we need to relabel so that a proper labelling is obtained and the structure of the components is preserved. If a has size k and b size $n-k$, then the number of ways to properly label the ordered pair (a, b) is $\binom{n}{k}$.

The symbolic method for EGFs

- EGFS are often used for labelled constructions. A labelled combinatorial object is one where each atom carries a positive integer label and all labels are distinct. The labelling is proper if the label set is $\{1, \ldots, n\}$.
- Example: a permutation is a properly labelled sequence of atoms.
- When forming the sum or product of labelled classes, we need to relabel so that a proper labelling is obtained and the structure of the components is preserved. If a has size k and b size $n-k$, then the number of ways to properly label the ordered pair (a, b) is $\binom{n}{k}$.
- Thus it makes sense to consider EGFs since

$$\left(\sum_n a_n z^n/n!\right)\left(\sum_n b_n z^n/n!\right) = \sum_n \left(\sum_k \binom{n}{k} a_k b_{n-k}\right) z^n/n!.$$

The symbolic method for EGFs

- EGFS are often used for labelled constructions. A labelled combinatorial object is one where each atom carries a positive integer label and all labels are distinct. The labelling is proper if the label set is $\{1, \ldots, n\}$.
- Example: a permutation is a properly labelled sequence of atoms.
- When forming the sum or product of labelled classes, we need to relabel so that a proper labelling is obtained and the structure of the components is preserved. If a has size k and b size $n-k$, then the number of ways to properly label the ordered pair (a, b) is $\binom{n}{k}$.
- Thus it makes sense to consider EGFs since

$$\left(\sum_n a_n z^n/n!\right)\left(\sum_n b_n z^n/n!\right) = \sum_n \left(\sum_k \binom{n}{k} a_k b_{n-k}\right) z^n/n!.$$

- Similarly, the EGF for sets is

The symbolic method - binary tree example

- A binary tree is either a single external node or an internal node connected to a pair of binary trees. Let $\mathcal{T}$ be the class of binary trees:

$$\mathcal{T} = \{ext\} \cup \{int\} \times \mathcal{T} \times \mathcal{T}.$$

In terms of a formal grammar

$$< tree >=< ext > | < int >< tree >< tree > .$$

The symbolic method - binary tree example

- A binary tree is either a single external node or an internal node connected to a pair of binary trees. Let $\mathcal{T}$ be the class of binary trees:

$$\mathcal{T} = \{ext\} \cup \{int\} \times \mathcal{T} \times \mathcal{T}.$$

In terms of a formal grammar

$$< tree >=< ext > | < int >< tree >< tree > .$$

- Give $< ext >$ weight a and $< int >$ weight b to obtain $T(z) = z^a + z^b T(z)^2$. Special cases: $a = 0, b = 1$ counts trees by internal nodes; $a = 1, b = 0$ by external nodes; $a = b = 1$ by total nodes.

The symbolic method - composition examples

- An integer composition is a sequence of positive integers. The size is the sum of the sequence. The counting OGF for the positive integers is $I(z) = z/(1-z)$, so the counting OGF for compositions by size is $(1 - z/(1-z))^{-1} = (1-z)/(1-2z)$ and there are 2^{n-1} compositions of n.

The symbolic method - composition examples

- An integer composition is a sequence of positive integers. The size is the sum of the sequence. The counting OGF for the positive integers is $I(z) = z/(1-z)$, so the counting OGF for compositions by size is $(1 - z/(1-z))^{-1} = (1-z)/(1-2z)$ and there are 2^{n-1} compositions of n.
- Let S be any subset of positive integers and let $I(z) = \sum_{n \in S} z^n$ be its counting OGF. Then $(1 - I(z))^{-1}$ enumerates compositions with parts restricted to S. Example: $S = \{1, 2, \ldots, r\}$ gives $(1-z)/(1-2z+z^{r+1})$.

The symbolic method - composition examples

- An integer composition is a sequence of positive integers. The size is the sum of the sequence. The counting OGF for the positive integers is $I(z) = z/(1-z)$, so the counting OGF for compositions by size is $(1 - z/(1-z))^{-1} = (1-z)/(1-2z)$ and there are 2^{n-1} compositions of n.
- Let S be any subset of positive integers and let $I(z) = \sum_{n \in S} z^n$ be its counting OGF. Then $(1 - I(z))^{-1}$ enumerates compositions with parts restricted to S. Example: $S = \{1, 2, \ldots, r\}$ gives $(1-z)/(1-2z+z^{r+1})$.
- Similarly we can count the number of terms in the sequence with another variable. The bivariate GF for compositions is

$$\frac{1}{1 - uz/(1-z)} = \frac{1-z}{1-z-uz} = \frac{1-z}{1-(1+u)z}.$$

The symbolic method - labelled tree example

- A properly labelled (unordered) tree is a connected acyclic graph with n vertices, each with one of the numbers $1, \ldots, n$.

The symbolic method - labelled tree example

- A properly labelled (unordered) tree is a connected acyclic graph with n vertices, each with one of the numbers $1, \ldots, n$.
- By symbolic method, the set $\mathcal{T}$ of rooted labelled unordered trees satisfies $\mathcal{T} = \{\bullet\} \times set(\mathcal{T})$ and so $T(z) = z \exp(T(z))$.

The symbolic method - labelled tree example

- A properly labelled (unordered) tree is a connected acyclic graph with n vertices, each with one of the numbers $1, \ldots, n$.
- By symbolic method, the set $\mathcal{T}$ of rooted labelled unordered trees satisfies $\mathcal{T} = \{\bullet\} \times set(\mathcal{T})$ and so $T(z) = z \exp(T(z))$.
- Lagrange inversion gives

$$a_n = \frac{n!}{n}[y^{n-1}] \exp(ny) = \frac{n!}{n}[y^{n-1}] \sum_k (ny)^k / k! = n^{n-1}.$$

The symbolic method - labelled tree example

- A properly labelled (unordered) tree is a connected acyclic graph with n vertices, each with one of the numbers $1, \ldots, n$.
- By symbolic method, the set $\mathcal{T}$ of rooted labelled unordered trees satisfies $\mathcal{T} = \{\bullet\} \times set(\mathcal{T})$ and so $T(z) = z \exp(T(z))$.
- Lagrange inversion gives

$$a_n = \frac{n!}{n}[y^{n-1}] \exp(ny) = \frac{n!}{n}[y^{n-1}] \sum_k (ny)^k / k! = n^{n-1}.$$

- Hence the number of labelled trees is n^{n-2}.

Coefficient extraction formulae (no proof)

Some basic results (best proved using the Cauchy integral formula):

- $$n! \approx \sqrt{2\pi n}(n/e)^n \qquad \text{(Stirling)}.$$

  Hence
  $$\frac{1}{n+1}\binom{2n}{n} \approx \frac{4^n}{\sqrt{\pi n}}.$$

Coefficient extraction formulae (no proof)

Some basic results (best proved using the Cauchy integral formula):

- $$n! \approx \sqrt{2\pi n}(n/e)^n \qquad \text{(Stirling)}.$$

  Hence
  $$\frac{1}{n+1}\binom{2n}{n} \approx \frac{4^n}{\sqrt{\pi n}}.$$
- Suppose $F(z) = \sum_n a_n z^n = G(z)/H(z)$ is rational and H has a unique root ρ of smallest modulus. Then
  $$a_n \approx C\rho^{-n} n^{p-1}$$
  where p is the order of ρ and C is easily computable.

Coefficient extraction formulae (no proof)

Some basic results (best proved using the Cauchy integral formula):

- $$n! \approx \sqrt{2\pi n}(n/e)^n \qquad \text{(Stirling)}.$$

  Hence
  $$\frac{1}{n+1}\binom{2n}{n} \approx \frac{4^n}{\sqrt{\pi n}}.$$
- Suppose $F(z) = \sum_n a_n z^n = G(z)/H(z)$ is rational and H has a unique root ρ of smallest modulus. Then
  $$a_n \approx C\rho^{-n}n^{p-1}$$
  where p is the order of ρ and C is easily computable.
- If $F(z) = z\phi(F(z))$ with $\phi(0) \neq 0$, then
  $$[z^n]F(z) = \frac{1}{n}[u^{n-1}]\phi(u)^n \quad \text{(Lagrange inversion formula)}.$$

Lagrange inversion example: degree-restricted trees

- Let $0 \in \Omega \subseteq \mathbb{N}$. We consider the class $\mathcal{T}_\Omega$ of all ordered plane trees such that the outdegree of each node is restricted to belong to Ω.

Lagrange inversion example: degree-restricted trees

- Let $0 \in \Omega \subseteq \mathbb{N}$. We consider the class $\mathcal{T}_\Omega$ of all ordered plane trees such that the outdegree of each node is restricted to belong to Ω.
- Examples: $\Omega = \{0, 1\}$ gives paths; $\Omega = \{0, 2\}$ gives binary trees; $\Omega = \{0, t\}$ gives t-ary trees; $\Omega = \mathbb{N}$ gives general ordered trees.

Lagrange inversion example: degree-restricted trees

- Let $0 \in \Omega \subseteq \mathbb{N}$. We consider the class $\mathcal{T}_\Omega$ of all ordered plane trees such that the outdegree of each node is restricted to belong to Ω.
- Examples: $\Omega = \{0, 1\}$ gives paths; $\Omega = \{0, 2\}$ gives binary trees; $\Omega = \{0, t\}$ gives t-ary trees; $\Omega = \mathbb{N}$ gives general ordered trees.
- Let $T_\Omega(z)$ be the enumerating GF of this class. The symbolic method immediately gives the equation

$$T_\Omega(z) = z\phi(T_\Omega(z))$$

where $\phi(x) = \sum_{\omega \in \Omega} x^\omega$.

Lagrange inversion example: degree-restricted trees

- Let $0 \in \Omega \subseteq \mathbb{N}$. We consider the class $\mathcal{T}_\Omega$ of all ordered plane trees such that the outdegree of each node is restricted to belong to Ω.
- Examples: $\Omega = \{0, 1\}$ gives paths; $\Omega = \{0, 2\}$ gives binary trees; $\Omega = \{0, t\}$ gives t-ary trees; $\Omega = \mathbb{N}$ gives general ordered trees.
- Let $T_\Omega(z)$ be the enumerating GF of this class. The symbolic method immediately gives the equation

$$T_\Omega(z) = z\phi(T_\Omega(z))$$

  where $\phi(x) = \sum_{\omega \in \Omega} x^\omega$.
- Lagrange inversion is tailor-made for this situation. We have

$$[z^n]T_\Omega(z) = \frac{1}{n}[u^{n-1}]\phi(u)^n.$$

Lagrange inversion example II - ternary trees

- Let $\Omega = \{0, 3\}$. Then the counting (by external nodes) OGF $T(z)$ satisfies $T(z) = z(1 + T(z)^3)$.

Lagrange inversion example II - ternary trees

- Let $\Omega = \{0, 3\}$. Then the counting (by external nodes) OGF $T(z)$ satisfies $T(z) = z(1 + T(z)^3)$.
- By Lagrange inversion we get

$$a_n = [z^n]T(z) = \frac{1}{n}[u^{n-1}]\left(1 + u^3\right)^n.$$

Lagrange inversion example II - ternary trees

- Let $\Omega = \{0, 3\}$. Then the counting (by external nodes) OGF $T(z)$ satisfies $T(z) = z(1 + T(z)^3)$.
- By Lagrange inversion we get

$$a_n = [z^n]T(z) = \frac{1}{n}[u^{n-1}]\left(1 + u^3\right)^n.$$

- By lookup we obtain

$$a_n = \begin{cases} \frac{1}{n}\binom{n}{k} & \text{if } n = 3k + 1 \\ 0 & \text{otherwise.} \end{cases}$$

Lagrange inversion example II - ternary trees

- Let $\Omega = \{0, 3\}$. Then the counting (by external nodes) OGF $T(z)$ satisfies $T(z) = z(1 + T(z)^3)$.
- By Lagrange inversion we get

$$a_n = [z^n]T(z) = \frac{1}{n}[u^{n-1}]\left(1 + u^3\right)^n.$$

- By lookup we obtain

$$a_n = \begin{cases} \frac{1}{n}\binom{n}{k} & \text{if } n = 3k + 1 \\ 0 & \text{otherwise.} \end{cases}$$

- Asymptotics are easily derived using Stirling's approximation.

Types of trees I

- A free tree is a connected graph with no cycles.
- A rooted tree is a free tree with a distinguished node called the root.
- An ordered tree is a rooted tree where the order of subtrees is important; recursively, a root connected to a sequence of ordered trees.
- A m-ary tree is an ordered tree where every node has 0 or m children.
- A labelled tree is a tree with n nodes such that each node is labelled by an element of $[n]$ and all labels are distinct.

Synonyms in literature: plane $=$ ordered; oriented $=$ rooted.

Some trees in analysis of algorithms

- Binary search tree: a binary tree with each internal node having a key, such that the key of each node n is $\leq$ all keys in R_n and $\geq$ all keys in L_n. Applications: database for comparable data, model for quicksort.
- Heap-ordered tree: a binary tree such that the key of each node is $\geq$ the key of anything in its subtree. Applications: priority queue.
- Trie: an m-ary tree where each external node may contain data; children of leaves must be nonempty. Applications: database for string data, model for radix exchange sort, leader election in distributed computing.

Tree attributes

- The size is the number of (external, internal, or just plain) nodes.
- The depth of a node in a rooted tree is the distance to the root.
- The maximum depth is the height. The sum of all depths of internal (external) nodes is the internal (external) path length.

Path length in binary trees (uniform model)

- The bivariate generating function $F(z,u)$ enumerating binary trees by number of nodes and internal path length satisfies the equation
$$F(z,u) = 1 + zF(zu,u)^2.$$
- The mean and variance are given by a standard computation. Note that
$$F_u(z,u) = 2zF(zu,u)[F_u(zu,u) + zF_z(zu,u)]$$
and so $F_u(z,1) = 2zF(z,1)[F_u(z,1) + zF_z(z,1)]$. Thus
$$\mu_n := \frac{[z^n]\frac{zF_z(z,1)}{1-2zF(z,1)}}{[z^n]F(z,1)}$$
- The mean μ_n is asymptotic to $\sqrt{\pi}n^{3/2}$, so the mean level of a node is of order $\sqrt{n}$. The variance is also of order $n^{3/2}$.

Path length in binary search trees

- Suppose we insert n distinct keys into an initially empty BST. The uniform distribution on permutations of size n induces the quicksort distribution on BSTs of size n.
- The internal path length equals the construction cost of a binary search tree of size n; dividing by n gives the expected cost of a successful search.
- Let $F(z,u) = \sum \frac{z^{|\pi|}}{|\pi|!} u^{\ell(\pi)}$ be the BGF of BSTs by size and internal path length. Note that $[z^n]F(z,u)$ is the PGF of internal path length on BSTs with n nodes.
  Then
  $$F_z(z,u) = F(zu,u)^2 \qquad F(0,u) = 1.$$
- Moments of the distribution are easily obtained as for the uniform model. The mean is $\sim 2n\log n$ and variance is in $\Theta(n^2)$. Quite a different shape.

String basics

- Let $\mathcal{A}$ be a finite set called the alphabet. A string over $\mathcal{A}$ is a finite sequence of elements of $\mathcal{A}$. The set of all strings over $\mathcal{A}$ is written $\mathcal{A}^*$.
- A subset of $\mathcal{A}^*$ is a language.
- If $\mathcal{A} = \{0, 1\}$ the strings are called bitstrings.
- Basic algorithmic questions: string matching (find a pattern in a given string); search for a word in a dictionary; compress a string. Many applications in computational biology, computer security, etc.

Hidden pattern occurrences

- The set of occurrences of the subsequence (hidden pattern) $-a_1 - a_2 - \cdots - a_k-$ in a string of length n corresponds to $\mathcal{A}^* a_1 \mathcal{A}^* a_2 \mathcal{A}^* a_3 \ldots \mathcal{A}^* a_k \mathcal{A}^*$.
- The counting OGF of $\mathcal{A}^*$ is $1/(1-mz)$ so the OGF for all pattern occurrences is $P(z) = z^k/(1-mz)^{k+1}$ where $m = |\mathcal{A}|$.
- The expected number of occurrences in a random "word" of length n is $[z^n]P(z)/(m^n) = m^{-k}\binom{n}{k}$.
- The OGF for total occurrences of the substring $a_1 a_2 \ldots a_k$ is $z^k/(1-mz)^2$ and a similar analysis applies.
- Note relevance to various conspiracy theories. We should expect a sufficiently long random text to contain any given hidden message.

Pattern avoidance

- We have already counted total occurrences of a given substring or pattern. Now we want to count number of words a_n not containing a given pattern (a harder problem).
- A nice trick: let T be the position of the end of the first occurrence of the pattern, X_n the event that the first n bits of a random bitstring do not contain the pattern. Then $S(z) = \sum_{n\geq 0} a_n z^n$ implies that

$$S(1/2) = \sum_{n\geq 0} a_n/2^n = \sum_{n\geq 0} \Pr(X_n) = \sum_{n\geq 0} \Pr(T > n) = E[T].$$

Pattern avoidance - simple example

- Given substring $\sigma = 00\cdots0$ of length k, let $S(z)$ be the counting OGF for bitstrings without σ as substring.
- Recursion/symbolic method gives

$$S(z) = \left(\sum_{i<k} z^i\right)(1 + zS(z))$$

so

$$S(z) = \frac{1 + z + z^2 + \cdots + z^{k-1}}{1 - z - z^2 - \cdots - z^k} = \frac{1 - z^k}{1 - 2z + z^{k+1}}.$$

- Asymptotics: $a_n \approx C\rho^{-n}$ where ρ is smallest modulus root of denominator.
- Note that $\rho = 1/2 + \rho^{k+1}/2$ and $0 < \rho < 1$. Thus $1/2 < \rho < 1/2 + 1/2^k$, etc, and we can compute ρ quickly by iteration.
- Note $S(1/2) = 2^{k+1} - 2$.

Substring patterns - autocorrelation polynomial I

- Consider an arbitrary binary string $\sigma = \sigma_0\sigma_1 \cdots \sigma_{k-1}$ of length k.

Substring patterns - autocorrelation polynomial I

- Consider an arbitrary binary string $\sigma = \sigma_0\sigma_1 \cdots \sigma_{k-1}$ of length k.
- For $0 \leq j \leq 1$, shift σ right j places. Define $c_j = 1$ if the overlap matches the tail $\sigma^{(j)}$ of σ, $c_j = 0$ otherwise. The autocorrelation polynomial is $c(z) = \sum_j c_j z^j$.

Substring patterns - autocorrelation polynomial I

- Consider an arbitrary binary string $\sigma = \sigma_0\sigma_1\cdots\sigma_{k-1}$ of length k.
- For $0 \leq j \leq 1$, shift σ right j places. Define $c_j = 1$ if the overlap matches the tail $\sigma^{(j)}$ of σ, $c_j = 0$ otherwise. The autocorrelation polynomial is $c(z) = \sum_j c_j z^j$.
- Let $\mathcal{S}$, (resp. $\mathcal{T}$) be the set of bitstrings not containing p (resp. containing it once at the end). Then

$$\mathcal{S} \cup \mathcal{T} \cong \{\epsilon\} \cup \mathcal{S} \times \{0,1\}$$
$$\mathcal{S} \times \{\sigma\} \cong \mathcal{T} \times \cup_{\{j:c_j \neq 0\}} \sigma^{(j)}$$

and the symbolic method gives $S(z) + T(z) = 1 + 2zS(z)$ and $S(z)z^k = T(z)c(z)$.

Substring patterns - autocorrelation polynomial II

- Thus

$$S(z) = \frac{c(z)}{z^k + (1-2z)c(z)}.$$

Substring patterns - autocorrelation polynomial II

- Thus

$$S(z) = \frac{c(z)}{z^k + (1-2z)c(z)}.$$

- Note $[z^n]S(z/2) = \Pr(X_n)$.

Substring patterns - autocorrelation polynomial II

- Thus
$$S(z) = \frac{c(z)}{z^k + (1-2z)c(z)}.$$
- Note $[z^n]S(z/2) = \Pr(X_n)$.
- Looking at the poles of $S(z/2)$ we see that $\Pr(X_n) \to 1$ exponentially fast as $n \to \infty$.

Substring patterns - autocorrelation polynomial II

- Thus
$$S(z) = \frac{c(z)}{z^k + (1-2z)c(z)}.$$
- Note $[z^n]S(z/2) = \Pr(X_n)$.
- Looking at the poles of $S(z/2)$ we see that $\Pr(X_n) \to 1$ exponentially fast as $n \to \infty$.
- Thus every fixed substring is contained with high probability in a sufficiently long string.

Regular languages

- Rational GFs always arise from the transfer matrix method.
- Special case: the counting GF of an unambiguous regular language is rational (Chomsky-Schützenberger, 1963).
- Recall that every regular language can be defined by an unambiguous regular expression.
- Thus if we construct a combinatorial class iteratively using only disjoint union, cartesian product, and sequence, the counting GF is rational.

Regular expression example

- Consider language (over alphabet $\{a, b\}$) defined by $(bb \mid a(bb)^*aa \mid a(bb)^*(ab \mid ba)(bb)^*(ab \mid ba))^*$ (number of b's is even, number of a's divisible by 3).
- The symbolic method gives

$$S(z) = \frac{(1-z^2)^2}{1-3z^2-z^3+3z^4-3z^5+z^6}.$$

  Hence $a_n \approx CA^n$, $A \cong 1.7998$.
- Need to check that the expression is unambiguous.

Patterns in strings: summary

- The generating function for any regular expression is a rational function and can be computed algorithmically.
- In certain special cases more efficient methods (such as autocorrelation polynomial) exist.
- We have really only considered the case where all letters appear independently and with equal probability. In real applications, letter probabilities vary and letters are not independent. The methods above can be extended to cope with this.

Tries

- Each binary tree corresponds to a set of binary strings (0 encodes left branch, 1 encodes right branch, string is given by labels on path to external node). This set of strings is prefix-free.
- Conversely a finite prefix-free set of strings corresponds to a unique binary tree, a full trie.
- More generally, we may stop branching as soon as the strings are all distinguished. This gives a trie, a binary tree such that all children of leaves are nonempty. Each string is stored in an external node but not all external nodes have strings. Can be described by symbolic method.
- A Patricia trie saves space, by collapsing one-way branches to a single node.
- Relevant parameters: number of internal nodes I_n; external path length L_n; length of rightmost branch R_n.

Trie recurrences

We assume that a trie is built from n infinite random bitstrings. Each bit of each string is independently either 0 or 1. We have

- $$L_n = n + \frac{1}{2^n}\sum_k \binom{n}{k}(L_k + L_{n-k})$$
$$I_n = 1 + \frac{1}{2^n}\sum_k \binom{n}{k}(I_k + I_{n-k})$$
$$R_n = 1 + \frac{1}{2^n}\sum_k \binom{n}{k} R_k.$$
- Let $L(z) = \sum_n L_n z^n/n!$, etc. Then
$$L(z) = 2L(z/2)e^{z/2} + ze^z - z;$$
$$I(z) = 2I(z/2)e^{z/2} + e^z - z - 1;$$
$$R(z) = R(z/2)e^{z/2} + e^z - z - 1;$$

Solving the trie recurrences, I

- If $\phi(z) = 2e^{z/2}\phi(z/2) + a(z)$, then by iteration we obtain

$$\phi(z) = \sum_{j\geq 0} 2^j e^{z(1-2^{-j})} a(2^{-j}z).$$

- Thus we obtain

$$L_n = n \sum_{j\geq 0} \left(1 - \left(1 - 2^{-j}\right)^{n-1}\right)$$

$$I_n = \sum_{j\geq 0} 2^j \left[1 - \left(1 - 2^{-j}\right)^n - \frac{n}{2^j}\left(1 - 2^{-j}\right)^{n-1}\right].$$

- How to derive an asymptotic approximation? See Flaj-Sedg p211, p402 for elementary arguments. Answers: $L_n \approx n \lg n, I_n \approx n/\lg 2$. More precise answers are obtained by complex methods (Mellin transform).

Solving the trie recurrences, II

- Define $\hat{\phi}(z) = e^{-z}\phi(z)$, etc (this is the Poisson transform). Then we have
$$\hat{\phi}(z) = 2\hat{\phi}(z/2) + \hat{a}(z).$$
- Iteration yields
$$\hat{\phi}(z) = \sum_{j\geq 0} 2^j \hat{a}(2^{-j}z).$$
- This gives, on inverting the transform,
$$L_n = \sum_{k\geq 2} (-1)^k \binom{n}{k} \frac{k2^{k-1}}{2^{k-1}-1}.$$
- Asymptotics for such alternating sums can be obtained by Rice's method.

Summary: tries

- A useful data structure for dictionary and pattern matching. Also a mathematical model for many algorithms.
- Asymptotically optimal $(\lg n)$ expected search cost.
- Space wastage: about 44% extra nodes $(1/\lg 2 - 1)$.
- Recurrences under the infinite random bitstring model yield GF equations that are tricky. Solution involves infinite sums of functions.
- Explicit formulae for solutions are infinite sums. Mellin transforms or Rice's integrals give precise asymptotics; elementary methods can also be used.

Randomized Algorithms

- Incorporating random choices into an algorithm can improve its expected performance. The key idea is that no adversary can force us into a bad case, because our choices are unknown.
- There are philosophical problems concerning random number generation on a deterministic computer, or even what "random" means. Pseudorandom generators are used in practice; their output passes most statistical tests for "randomness".
- We can also think of a randomized algorithm as an element randomly chosen from a set of algorithms.
- There are two main types: Monte Carlo (answer may be wrong, running time is deterministic) and Las Vegas (answer is correct, running time is random).

Monte Carlo algorithms

- If the runtime on a given instance is deterministic and the answer is correct with bounded probability, we have a Monte Carlo algorithm.
- A MC algorithm for a decision problem is biased if one of the answers (yes/no) is always correct when given. In other words, for example, $P(Y|N) \equiv$ P(says Y given answer is N) $= 0, P(N|N) = 1, P(Y|Y) = p, P(N|Y) = 1 - p$.
- Examples: fingerprinting (verification of identities); primality testing.

Fingerprinting

- Suppose we have a set U and want to determine whether elements u, v are equal. It is often easier to choose a fingerprinting function f and compute whether $f(u) = f(v)$, in which case we return yes. Such methods are always biased: $P(N|Y) = 0$.
- Example: X, Y, Z are $n \times n$ matrices, and we want to know whether $XY = Z$. Choose a random n bit vector r and compute XYr and Zr. Can show $P(Y|N) \leq 1/2$.
- Example: M is a symbolic matrix in variables $x_1, \ldots x_n$; we want to know whether $\det(M) = 0$. Choose a finite subset S of $\mathbb{C}$ and choose $r_1, \ldots r_n$ independently and uniformly from S. Substitute $x_i = r_i$ for all i and compute the determinant. Can show $P(Y|N) \leq m/|S|$ where m is the total degree of the polynomial $\det(M)$.
- There are many specific applications of the above examples.

Improving biased Monte Carlo algorithms

- Let $0 < p < 1$. Say a MC algorithm is p-correct if its error probability is at most $1 - p$ on every instance (sometimes p depends on the size, but never on the instance itself).
- This means $P(Y|N) \leq 1 - p, P(N|Y) \leq 1 - p$.
- If, say, NO is always right then we can improve our confidence in the answer by repeating the algorithm n times on the same instance. This is amplification of the stochastic advantage. If we ever get NO, report NO. Else report YES. Probability of error is at most $(1 - p)^n$. To reduce this to ε requires number of trials proportional to $\lg(1/\varepsilon)$ and to $-\lg(1 - p)$.

Improving unbiased Monte Carlo algorithms

- We need $p \geq 1/2$. Repeat n (odd) times and return the more frequent answer. Analysis is more complicated.
- Let $X_i = 1$ if ith run gives correct answer, 0 otherwise. Then X_i is a Bernoulli random variable and $X = \sum_{i=1}^{n} X_i$ is binomial with parameter p. Probability of error of repeated algorithm is $P(X < n/2)$. This is just $\sum_{j<n/2} \binom{n}{j} p^j (1-p)^{n-j}$.
- Simplifying this could be done, but it is easier to use the normal approximation: X is approximately normal with mean np and variance $np(1-p)$ for n large enough. Use table of normal distribution to work out size of n for given ε. Answer is proportional to $\lg(1/\varepsilon)$ and $(p-1/2)^{-2}$.

Primality testing

- We wish to know whether a given positive integer n is prime (has no proper factor other than 1).
- Fermat (1640) proved that if n is prime, and $1 \leq a \leq n-1$, then $a^{n-1} \equiv 1 \mod n$. However if n is composite (not prime) then this equality may or may not hold.
- An obvious idea is: given n, choose a randomly and compute $a^{n-1} \mod n$ (recall that we can do this quickly, using a divide-and conquer approach). If the answer is not 1, we report NO (such an a is called a witness).
- This gives a biased Monte Carlo algorithm with $P(N|N) = 1, P(Y|N) = 0$. What about $P(N|Y), P(Y|Y)$?
- Unfortunately $P(N|Y)$ can be made arbitrarily close to 1 (some n have a lot of false witnesses). So this algorithm is not p-correct for any $p > 0$.

Improving the primality testing algorithm

- There is a more complicated test (Miller-Rabin, 1976) based on Fermat's result that gives $P(N|Y) \leq 1/4$, and hence a $3/4$-correct algorithm.
- There is an even more complicated test (Agrawal-Kayal-Saxena 2002) that is in fact always correct, and this gives the first worst-case polynomial-time algorithm for primality. But it is not as fast as the randomized algorithm above.

Las Vegas algorithms

- Nice properties: can find more than one solution even on same input; breaks the link between input and worse-case runtime.
- Every Las Vegas algorithm can be converted to a Monte Carlo algorithm: just report a random answer if the running time gets too long.
- Examples:
  - randomized quicksort and quickselect;
  - randomized greedy algorithm for n queens problem;
  - integer factorization;
  - universal hashing;
  - linear time MST.

Improving Las Vegas algorithms

- If a particular run takes too long, we can just stop and run again on the same input. If the expected runtime is fast, it is very unlikely that many iterations will fail to run fast.
- To quantify this: let p be probability of success, s the expected time to find a solution, and f the expected time when failure occurs. Then expected time t until we find a solution is given by $t = ps + (1-p)(f+t)$, so $t = s + (1-p)f/p$.
- We can use this to optimize the repeated algorithm.